权大师Redis入门培训文档

[1.了解redis(掌握) 7](#_Toc1808193428)

[1.1.redis简介 8](#_Toc2145646295)

[1.2.redis数据结构 8](#_Toc989531183)

[1.3.系统支持 9](#_Toc619883405)

[1.4.其他介绍 9](#_Toc853572282)

[1.5.redis安装 11](#_Toc885061592)

[1.6redis常用工具 14](#_Toc886445490)

[1.7redis服务器启动 15](#_Toc1170527650)

[1.8redis客户端 17](#_Toc196167455)

[1.9热配置修改 19](#_Toc478381361)

[1.10多数据库 20](#_Toc1264629482)

[2.常用命令(掌握) 21](#_Toc1722996473)

[2.1 keys \* 21](#_Toc1625438162)

[2.2 flushall 21](#_Toc47770148)

[2.3根据规则获取key列表 22](#_Toc225496870)

[2.4redis命令不区分大小写 22](#_Toc628397583)

[2.5判断key是否存在 23](#_Toc1607485518)

[2.6删除key 23](#_Toc1533168652)

[3.string(掌握) 24](#_Toc283371144)

[3.1数字递增 25](#_Toc1144869204)

[3.2赋值和取值 26](#_Toc2088111764)

[3.3命名空间 26](#_Toc1135305638)

[3.4指定增加步长 26](#_Toc143307722)

[3.5减少步长 27](#_Toc1915556615)

[3.6增加浮点数步长 27](#_Toc2105117000)

[3.7字符串追加 28](#_Toc1142885247)

[3.8设置多个key-value 29](#_Toc1540771980)

[4.hash(掌握) 29](#_Toc564913631)

[4.1赋值和取值 30](#_Toc722239867)

[4.2批量赋值取值 31](#_Toc2048812763)

[4.3获取某个key下所有数据 32](#_Toc1093655607)

[4.4判断字段是否存在 32](#_Toc382949647)

[4.5hsetnx赋值 33](#_Toc2046975410)

[4.6增加数字 33](#_Toc2083186790)

[4.7删除属性 34](#_Toc1002833052)

[4.8字符串和散列 35](#_Toc753064044)

[4.9特有命令 36](#_Toc820764735)

[4.10字符串和离散命令对比 36](#_Toc1889278543)

[5.list(掌握) 37](#_Toc1923591694)

[5.1左侧或右侧添加数据 37](#_Toc1016932190)

[5.2左右两侧弹出数据 38](#_Toc220176256)

[5.3获取list长度 38](#_Toc1040737529)

[5.4获取list片段 39](#_Toc592445015)

[5.5删除list指定值 40](#_Toc1845614418)

[5.6设置或获取索引处数据 40](#_Toc1088507677)

[5.7list截取 42](#_Toc817941885)

[5.8根据给定的value插入数据 43](#_Toc326528353)

[5.9将数据从a列表转移到b列表 43](#_Toc548509547)

[5.10模拟队列和栈操作 44](#_Toc203626889)

[5.11list-hash-string 45](#_Toc609899497)

[6.Set--待处理 45](#_Toc1693378752)

[7.Zset--待处理 46](#_Toc144255005)

[8.redis事务(理解) 46](#_Toc1745205136)

[8.1事务语法 46](#_Toc1836686474)

[8.2错误处理 47](#_Toc2059811620)

[8.3 watch 49](#_Toc1702838488)

[8.4 unwatch 51](#_Toc832088073)

[9.失效时间(掌握) 53](#_Toc1453099953)

[9.1使用场景 53](#_Toc120268471)

[9.2失效设置 54](#_Toc1554327940)

[9.3查看失效剩余时间 55](#_Toc1354429068)

[9.4取消失效时间 56](#_Toc1213924078)

[9.5失效命令对比 57](#_Toc1937277588)

[9.6访问频率限制 58](#_Toc1253920831)

[9.7设置redis使用最大内存 58](#_Toc1149627221)

[10.消息队列(掌握) 59](#_Toc792626992)

[10.1点对点实现 59](#_Toc2006984875)

[10.2发布/订阅实现 61](#_Toc1970391956)

[10.3发布者 62](#_Toc534421887)

[10.4订阅者 62](#_Toc1783092922)

[10.5取消订阅 64](#_Toc839840498)

[10.6规则订阅 65](#_Toc754598143)

[10.7规则退订 66](#_Toc676346803)

[11.管道(理解) 67](#_Toc1432285513)

[11.1为何使用管道 67](#_Toc452728913)

[11.2节省空间 68](#_Toc1764854480)

[12.持久化(理解) 68](#_Toc102743750)

[12.1RDB模式 68](#_Toc779257266)

[12.2RDB快照过程 70](#_Toc165880380)

[12.3RDB总结 71](#_Toc306370639)

[12.4AOF模式 71](#_Toc1389156764)

[12.5AOF总结 71](#_Toc1859259132)

[13.复制(理解) 72](#_Toc450625644)

[13.1复制意义 73](#_Toc986878252)

[13.2复制配置 73](#_Toc1548461958)

[13.3如何实现复制 73](#_Toc362953617)

[13.4.复制原理 76](#_Toc542233092)

[13.5读写分离 77](#_Toc233066383)

[13.6.从库持久化 77](#_Toc1816053570)

[14.安全(掌握) 78](#_Toc662501563)

[14.1绑定安全IP 78](#_Toc1787394323)

[14.2数据库密码 78](#_Toc1022998990)

[14.3.命令重命名 80](#_Toc1876425642)

[15.通讯协议(了解) 80](#_Toc1577188263)

[15.1简单协议 81](#_Toc129436173)

[15.2.统一请求协议 81](#_Toc878569215)

[16.redis客户端介绍 82](#_Toc222331608)

[16.1查看耗时命令(掌握) 82](#_Toc2136421049)

[16.2命令监控(掌握) 83](#_Toc701477523)

[16.3phpRedisAdmin(了解) 83](#_Toc756753495)

[16.4RDBtools(掌握) 84](#_Toc1772030323)

文档修改记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 修改内容描述 | 修改人 | 日期 | 备注 |
| 001 | 建立 | 张洪图 | 2017-03-29 |  |
| 002 | 完成 | 张洪图 | 2017-04-16 | 配置文件不生效不知道如何处理 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 1.了解redis(掌握)

## 1.1.redis简介

redis（remote dictionary server)是远程服务字典的缩写、redis是一种Nosql技术、开源的高级key-value存储系统。

redis和memcached比较，两者最大的不同在于后者不提供持久化的数据保存机制，而redis可以将数据保存在磁盘中。

redis在运行过程中，数据存放在内存中，性能极高，因此对内存要求比较高。

redis是单线程服务的，只有一个线程。

简介参考：<http://www.tuicool.com/articles/36Rzamv>

## 1.2.redis数据结构

思考：redis有哪些数据类型呢？

你能想到哪些，和你所用的编程语言中哪些数据结构有相似之处呢？

数据类型

1. 字符串(string)，编程中的数字、字符串等信息，都是用字符串处理。等价于java中的数字和字符串。
2. 散列(hash)，等价于编程中的map，类似于二叉树。
3. 列表(list)，等价于编程中的list，按照索引有序数据、数据可以重复。java中等同于LinkedList。
4. 集合(set),等价于编程中set，数据不允许重复，数据无序排列。
5. 有序集合(zset),数据不允许重复，且数据有序。

所以简单记起来如下：

字符串、散列、列表、集合、有序集合

## 1.3.系统支持

redis对Linux和mac系统支持性良好，但是对window环境支持比较差，如果学习和使用都尽可能使用Linux和mac，而不要使用window系统。

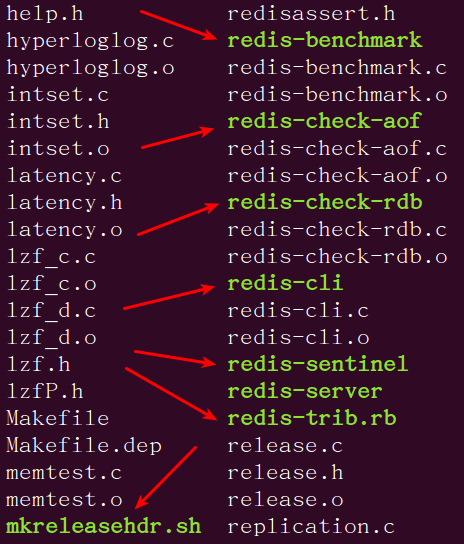
## 1.4.其他介绍

redis代码仅仅3万多行，适合进行源码阅读。虽然是C语言编写，但如果对某一门语言语法精通或者熟练掌握，那么学习另一门语言的语言很容易的。同时我对redis源码做一次讲解，做源码阅读探索，进行源码阅读技巧的学习和分享。

借鉴红警的术语：redis常用工具大本是src目录。



redis如下：



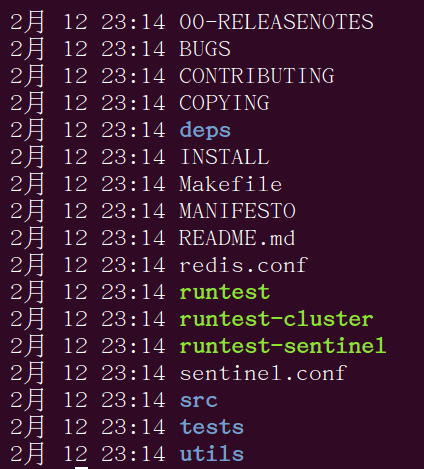
## 1.5.redis安装

下载redis源码包，目前版本redis-3.2.8.tar.gz

redis官网：<https://redis.io/>

解压缩命令：tar zxvf redis-3.2.8.tar.gz

解压后的redis-3.2.8如下：



简单介绍若干文件和目录

[个人英文一般，如果哪块翻译错误见谅，也请及时对我的文档进行修改]

1. you should read readme.md first,你要做的第一件事是务必阅读readme.md，介绍redis操作都是使用readme.md中内容。囫囵吞枣地阅读下来，收获也会颇丰。

redis数据结构介绍：

<https://redis.io/topics/data-types-intro>

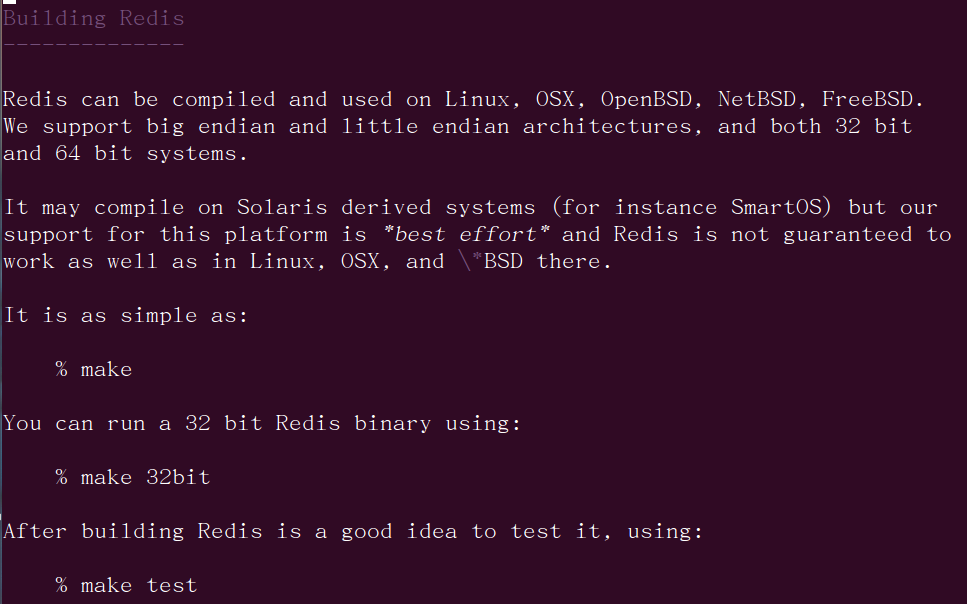
redis练习网页：如果没有本地环境，可以在此练习

<http://try.redis.io>

最权威的命令列表：

<https://redis.io/commands>

解压缩后如何编译redis



[题外话：个人建议尝试以官方文档为主，实在不明白再去百度，这样对个人能力是个很好的提升，英文长久不使用而感到烦闷而已，多用用和使用筷子一样]

这块介绍如何编译redis，即解压redis-3.2.8.tar.gz,然后在文件夹内进行编译，如果是64位系统，直接使用make命令。

编译后如何使用redis呢？



这些内容同时存在于redis的github介绍中：

<https://github.com/antirez/redis/>

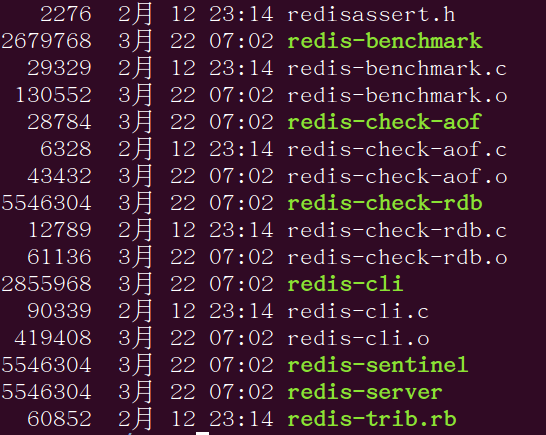
1. redis.conf，对redis服务器的所有参数化配置在此。

简单介绍若干，其他内容大部分会在后面所有涉猎到。

具体请查看Redis配置文件详解，或者参考：

<http://blog.csdn.net/Jack__Frost/article/details/67633975>

## 1.6redis常用工具



1. redis-server，redis服务器
2. redis-cli，redis客户端
3. redis-banchmark，redis性能测试工具

4.redis-check-aof，aof文件修复工具

5.redis-check-dump，rdb文件检查工具

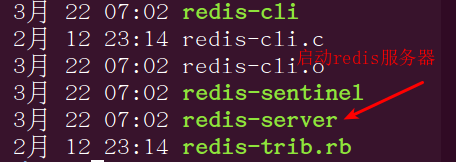
6.redis-trib.rb,管理redis集群工具,ruby，1600多行code

7.redis-sentinel，集群管理工具，监控节点，修复节点故障，提高集群可用性。

## 1.7redis服务器启动

参考1.6redis常用工具

步骤：redis目录下的src执行redis-server

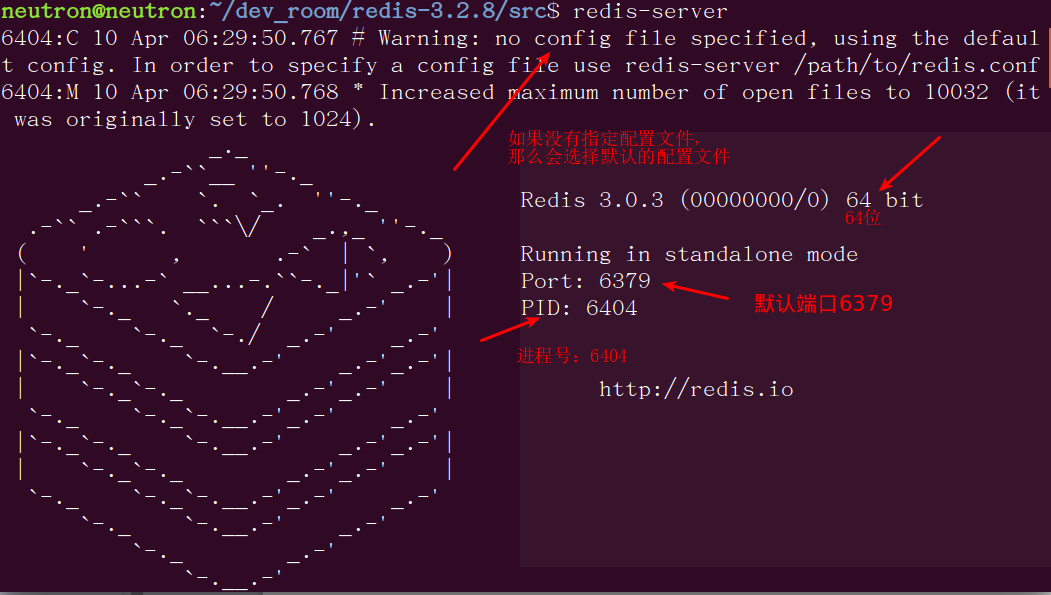


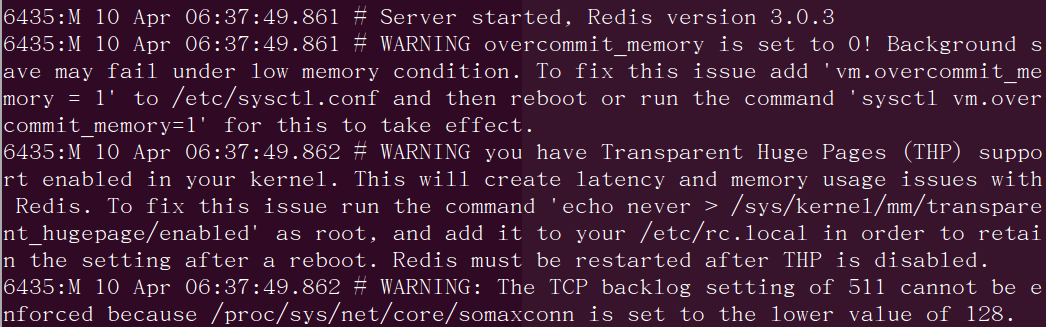
目前我的redis目录：

/home/neutron/dev\_room/redis-3.2.8/src

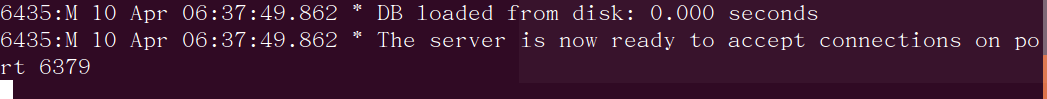
执行命令：

redis-server





以上内容未做介绍,研究后补上。

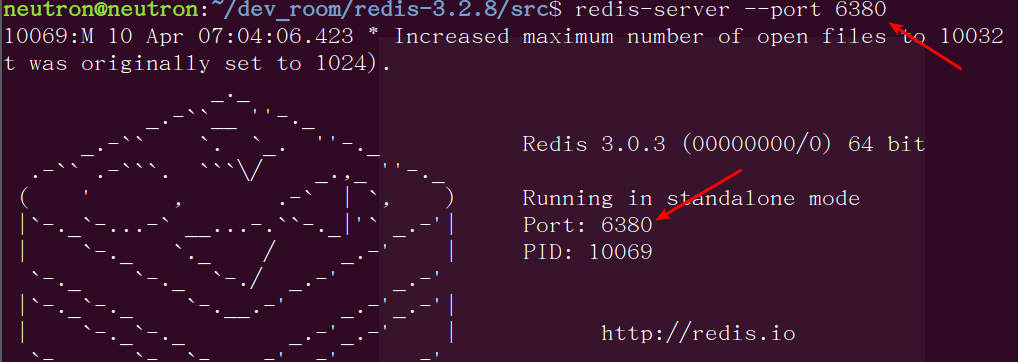


Redis启动时会从硬盘上加载已经持久化的数据，从硬盘上加载数据0秒；服务器已经对6379端口进行监听，已经可以正常建立redis客户端和服务器连接。

自定义端口进行redis启动

自定义端口使用，redis-server --port 自定义端口号

eg：redis-server --port 6380

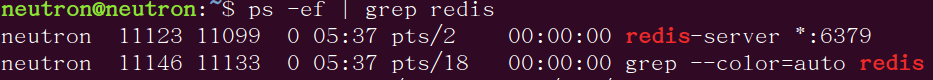


编写启动脚本，参考utils下redis\_init\_script文件

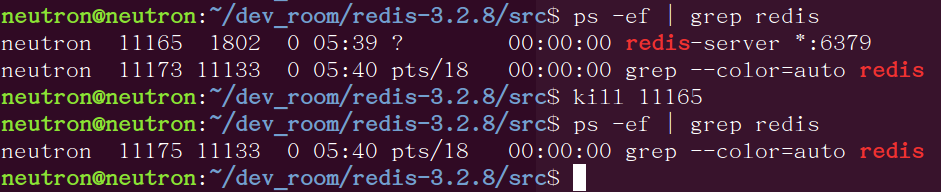
1.8redis服务器停止

正确停止redis操作，向redis发送shutdown命令

ps -ef | grep redis



redis-cli -p 6379 shutdown

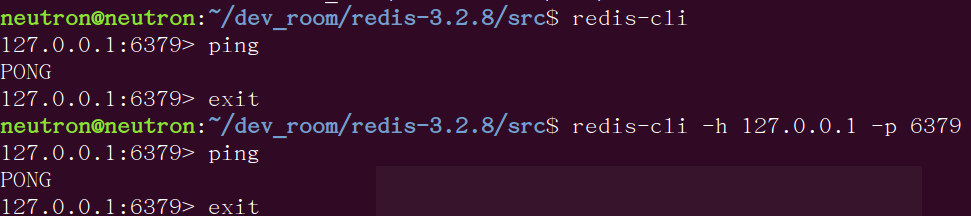


kill redis进程pid效果等同于shutdown

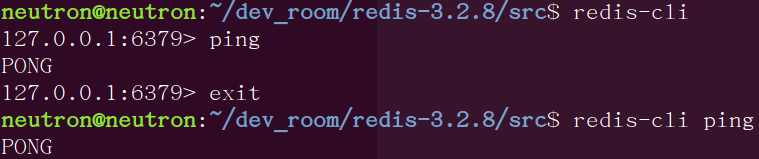
## 1.8redis客户端

连接服务器：redis-cli -h 服务器地址 -p 端口

如果服务器ip是127.0.0.1，端口是6370，那么可以直接使用redis-cli

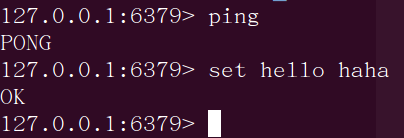


检测客户端和服务器是否正常连接，使用ping命令，返回pong表示连接正常。以下两种方式都可以：



1.9命令返回值

状态回复：显示状态信息表示操作、连接成功，比如OK，PONG



错误回复：以error打头



整数返回值，数据操作时



字符串回复，获取数据使用双引号引上，如果不存在返回nil

正常返回：

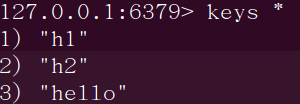


数据不存在：返回nil



单词nil表示无值，并非缩写。

5.多行字符串，比如keys \*



## 1.9热配置修改

什么含义？运行时修改配置文件内容，修改后会直接生效，不需要重新编译后才能使用，相当于热加载。

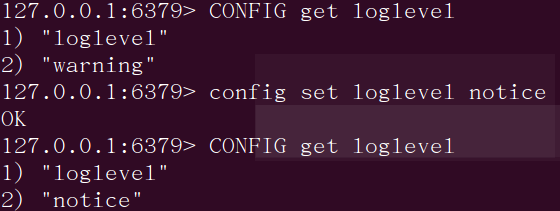
格式：config set configName value

eg：运行时设置日志级别 config set loglevel warning

获取配置

格式:config get configName

eg：config get loglevel



注意：并不是所有配置都可以运行时进行修改，所以在确认修改某个配置时，请确认运行时修改是否可以生效。

## 1.10多数据库

[参考redis配置文件详解中04.日志信息中databases说明]

redis默认16个数据库，索引0到索引15；可以在配置文件databases来修改个数；客户端和服务器连接默认选择第0个数据库；使用select来进行数据库切换，例如选择索引为4的数据库使用select 4；不支持数据库重命名，因此必须牢记哪个库存放哪些数据；不支持为每个数据库设置密码，支持对所有数据库设置同一密码。例如古代会给城池设置大门设置兵士进行驻守，但不会给每个百姓家设置城门和士兵驻守。或者每家都会设置一道防盗门，但是每个房间不会单独在设置防盗门；每个空的redis实例只占用内存1M左右；

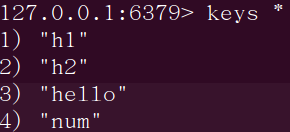
以上就是了解redis的内容，需要知道的redis数据结构、适合redis的系统、redis安装、redis常用工具、redis客户端使用、热配置修改、多数据库

# 2.常用命令(掌握)

## 2.1 keys \*

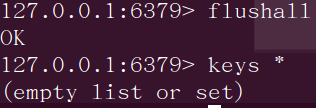
查看所有key值，keys \*.

如果redis存储大量数据，执行keys \*属于找死的节奏。



## 2.2 flushall

flushall，清空某个实例中所有数据库中的数据。这是找死的命令，数据量大的时候，将该命令进行重命名，禁止执行。



还有flushdb，清空某个实例中某个索引的数据库中的数据。

## 2.3根据规则获取key列表

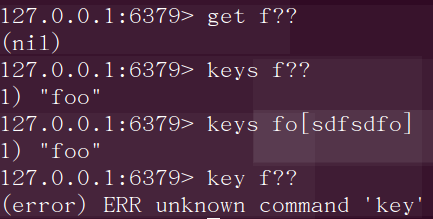
格式：keys [pattern]

规则：pattern规则：？表示匹配一个字符，\*表示匹配任意字符，[]表示匹配括号间任一字符[和我们使用的正则表达式作用一样]

案例：

keys \* 获取redis中所有键，如果key很多，会影响性能.

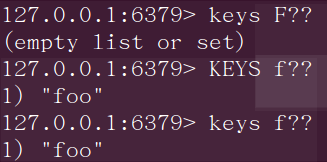
keys f?? -> foo, keys fo[asfo] -> foo



注意：使用的是keys，不是key。

## 2.4redis命令不区分大小写

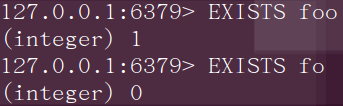
redis命令不区分大小写，但key和value区分大小写。



## 2.5判断key是否存在

判断key是否存在

定义：exists key



返回值：存在key返回1，不存在key返回0

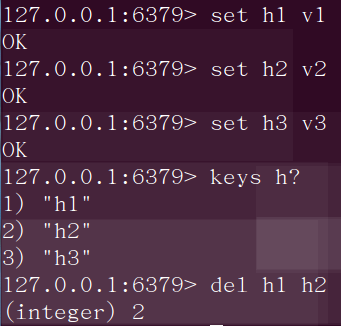
说明：redis使用c语言编写，在c中没有true和false，在c中使用0表示false，非0表示true。在redis中所有的true和false，分别是1和0；

## 2.6删除key

作用：删除key

定义：del key [key ...]

返回值：删除key的个数



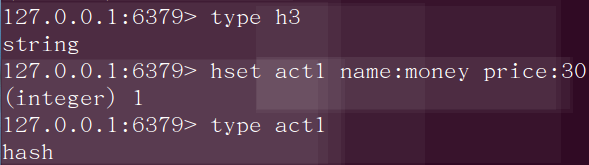
说明：一般而言redis删除多条数据时返回值是删除数据的个数。

2.7获取value数据类型

数据类型就是redis中的value数据类型。

作用：获取值的数据类型

定义： type key



如果没有key值存在，会返回none



# 3.string(掌握)

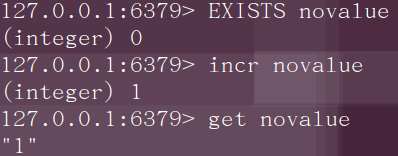
redis数据结构基础string，字符串主要存储的数据是序列化的对象、数字、字符串内容，最大存储空间是512M。

## 3.1数字递增

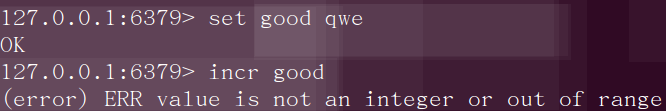
前提：key对应的value是整数类型

格式：incr key

说明：如果key不存在，那么该key对应的value的默认值设置为0后进行加1操作；如果key存在，而对应的value不是整数，那么会报错。



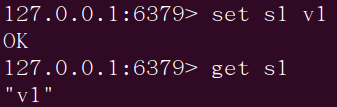
错误数据提示：



redis命令是原子性操作，因此不需要考虑并发操作导致的问题。

incr使用场景：排行榜、计数器、点赞、生成主键Id、访问量等需要进行计数的场合。

## 3.2赋值和取值



赋值：set key value

取值：get key

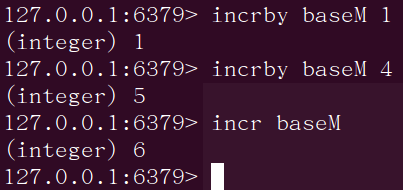
## 3.3命名空间

命令空间规则：对象类型:对象Id:对象属性

目前场景命名：业务模块:具体业务:作用:数据:key类型

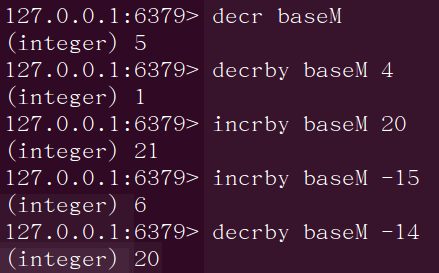
其中命名空间可读性第一，空间节省是次要信息。

## 3.4指定增加步长



格式：incrby key increment

## 3.5减少步长

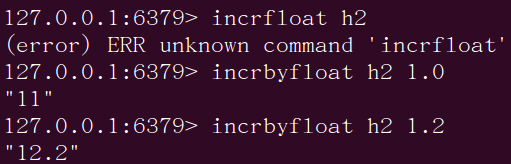


通过案例可知：减少指定步长：decrby key decrement

减少步长：decr key

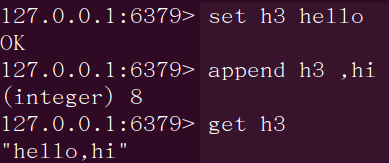
减少负数指定步长等于增加指定步长，增加负数步长等于减少指定步长。

## 3.6增加浮点数步长



参考案例:没有incrfloat，只有incrbyfloat

## 3.7字符串追加



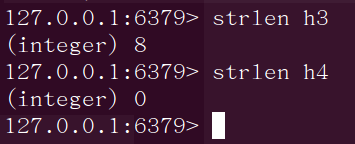
向尾部追加值: append key value

返回值:字符串长度.

append foo .meet -> 6

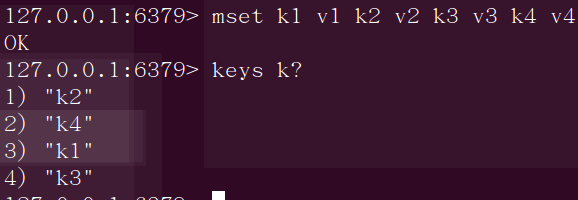
get foo -> 4.meat

3.8字符串长度

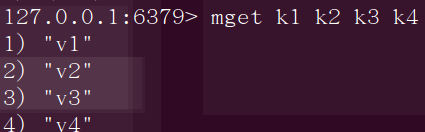


格式：strlen key

## 3.8设置若干key-value



设置多个键值对:mset key value[key value ...]



获取多个值: mget key[key ...]

字符串总结：字符串是redis最常用的数据结构，良好的命名空间是根本，防止出现数据覆盖现象。不遵循一定的命名空间出现数据覆盖情况，问题很难定位。常用在验证码、计数、少量数据缓存情景。

# 4.hash(掌握)

Hash的作用是存储字段和字段值的映射。其中字段值只能是字符串，不支持其他数据类型，不支持嵌套类型。存储时使用对象类别和Id作为key，字段表示属性、字段值存储属性值，hash是存储对象的完美解决方案，比如java自定义的对象。

hash不规定每个结构必须有哪些字段和数据，可以类似nosql自动给每条数据进行扩充或者削减字段。

注意：hash的操作命令都是以h开头，即hash的第1个字母。

## 4.1赋值和取值

赋值操作：hset key field value

取值操作：hget key field

[字符串赋值取值：set key value；get key value]



如果key不存在，返回nil；

key存在而field不存在，返回nil。

|  |  |  |
| --- | --- | --- |
| string | set key value | get key |
| hash | hset key field value | hget key field |

类比：string是进入学校直接获取东西，hash是进入学校后同时要进入某个教室去获取东西。

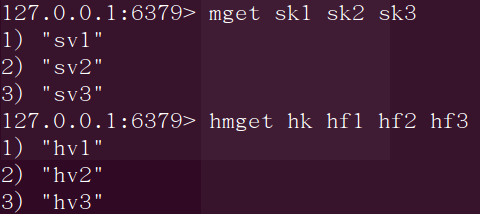
## 4.2批量赋值取值

|  |  |  |
| --- | --- | --- |
| string | mset [key value ..] | mget [key ..] |
| hash | hmset key [field value..] | hget key [field..] |

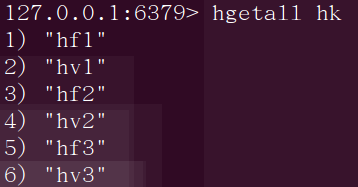
赋值操作



取值操作

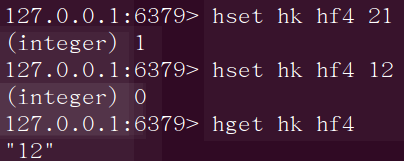


## 4.3获取某个key下所有数据

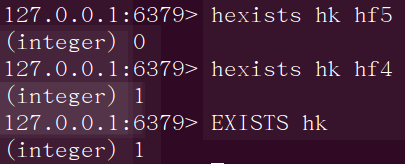


格式：hgetall key

注意：hset不区分插入和修改，插入操作去创建返回1，更新操作返回0，key不存在时自动创建。



## 4.4判断字段是否存在

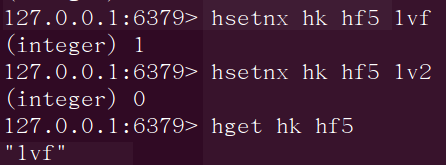


key或field不存在返回0，key field存在返回1

|  |  |
| --- | --- |
| all | exists key |
| hash | hexists key field |

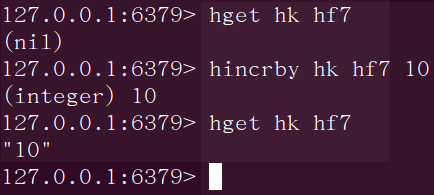
任意类型判断key是否存在使用exists；判断hash的某个field是否存在hexists key field。

## 4.5hsetnx赋值



定义：hsetnx key field value，字段存在不执行操作，字段不存在执行操作。其中hsetnx中的nx表示not exists的含义。

## 4.6增加数字



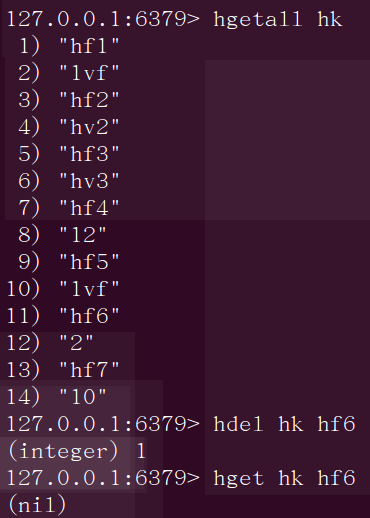
定义: hincrby key field increment，没有hincr命令.属性不存在会进行创建后赋值，属性如果不是整数会报错。

|  |  |  |
| --- | --- | --- |
| string | incr key | incrby key |
| hash | 无 | hincrby key field |

## 4.7删除属性

|  |  |  |
| --- | --- | --- |
| string | del key | 无 |
| hash | del key | hdel key [field..] |

删除key相同，删除属性hdel key [field..]



## 4.8string和hash

格式：hash的命令增加h，即hash的首字母；

存储：hash适合存储自定义对象，不建议将对象进行序列化后保存字符串。缺点是每次都要将整个字符串数据获取出来然后反序列化获取数据等等，尤其对频繁操作对象属性时。

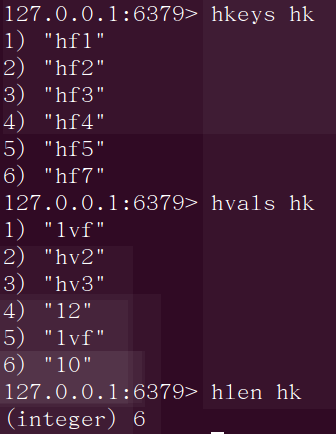
灵活性：hash可以灵活处理属性值，而字符串必须全部拿出来后进行操作。

复杂度：字符串只适合简单业务场景中，但是简单业务场景使用最频繁；hash适合比较复杂的业务场景中，尤其适合在网站的各类活动中。

## 4.9特有命令

|  |  |  |
| --- | --- | --- |
| 获取属性名 | hkeys key | Eg：hkeys hk |
| 获取属性值 | hvals key | Eg：hvals hk |
| 获取属性个数 | hlen key | Eg：hlen hk |

注意：hk是key值



## 4.10string和hash对比

|  |  |  |
| --- | --- | --- |
|  | string | hash |
| 赋值 | set key value | hset key field value |
| 取值 | get key | hget key field |
| 批量赋值 | mset [key value ..] | hmset key [field value..] |
| 批量取值 | mget [key..] | hmget key [field ..] |
| 自增 | incr key | 无 |
| 步长自增 | incrby key step | hincrby key field step |
| 查询key | keys pattern | hkeys key |
| 删除数据 | del key | del key field |
| 获取长度 | strlen key | hlen key（field个数） |

# 5.list(掌握)

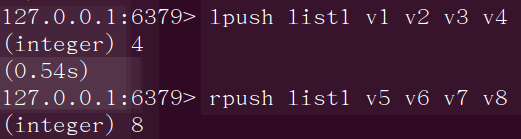
redis中的list相当于java中LinkedList，便于两边插入和删除，不利于遍历数据，通过索引访问数据比较慢，适合双向链表的操作。

使用场景：记录日志、消息队列、获取最新的数据而忽略旧数据，比如top推荐、top排名。

## 5.1左侧或右侧添加数据

左侧添加数据 lpush key value[values ...]

右侧添加数据 rpush key value[values ...]



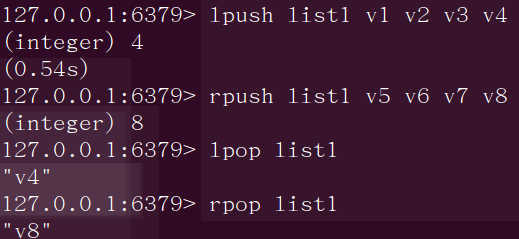
返回值是list的长度

结果：[v4 v3 v2 v1 v5 v6 v7 v8]

## 5.2左右两侧弹出数据

左侧弹出数据：lpop key

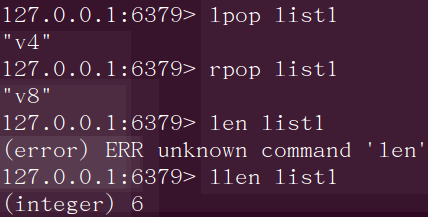
右侧弹出数据：rpop key



lpop和rpop，从列表中删除数据，返回值就是移除的数据值

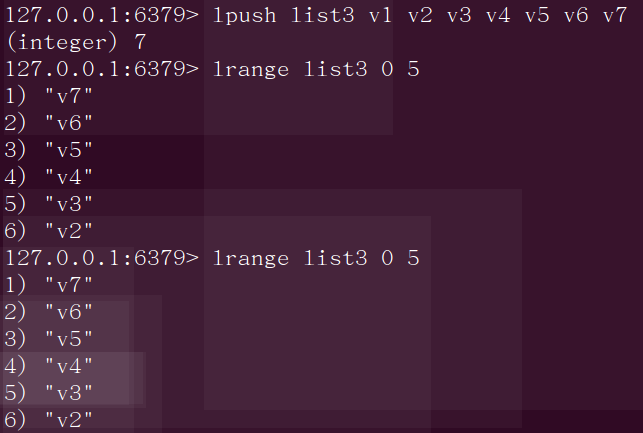
## 5.3获取list长度

承接5.2弹出数据后的list长度



key不在返回0，时间复杂度o（1）

## 5.4获取list片段



获取列表片段，lrange key start stop，包括两端索引位置数据，获取数据但是不会pop。redis的索引和其他语言一样，从0开始。

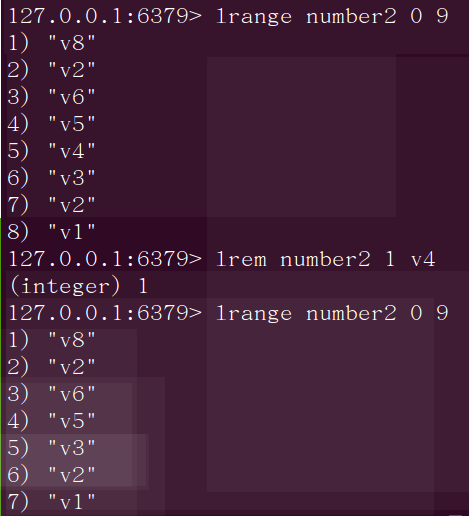
第一步：lpush后结果是[v7、v6、v5、v4、v3、v2、v1]

第二步：遍历数据结果v7、v6、v5、v4、v3、v2

第三步：lrange数据后数据并没有删除。

## 5.5删除list指定值

删除列表中指定值 lrem key count value，删除前count个值为value的元素，返回值是删除个数。

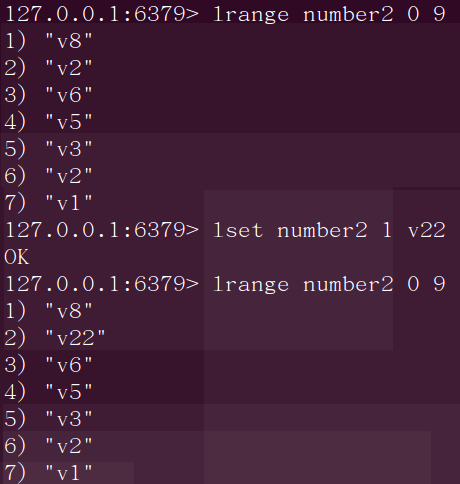


删除的是左面或右面数几个value。

## 5.6设置或获取索引处数据

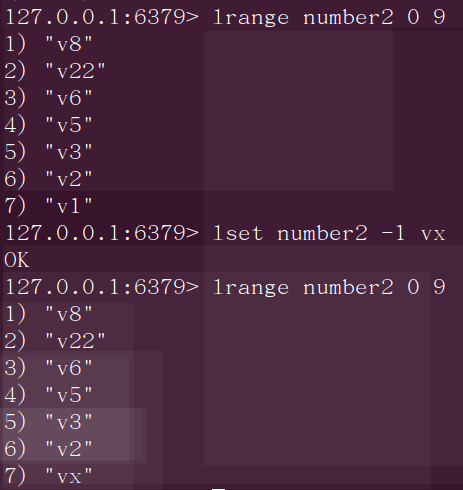
设置特定索引的值 lset key index value

eg: lset list1 3 index3



注意：没有rset操作，如果打算从右侧设置数据，那么index写成负数，右边第一个是-1，以此类推。

redis和java中list一样，都支持从右边操作数据，可以使用负整数，比如lindex list1 -2。

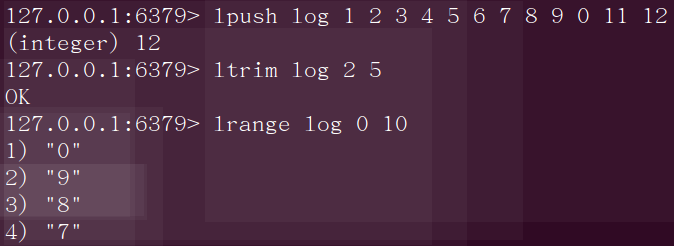


## 5.7list截取

列表截取，删除指定索引之外的数据。

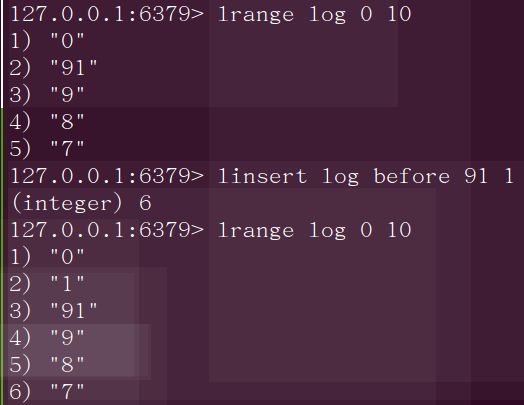
格式：ltrim key start stop。

比如只保留前100条log，ltrim log 0 99



## 5.8根据给定的value插入数据

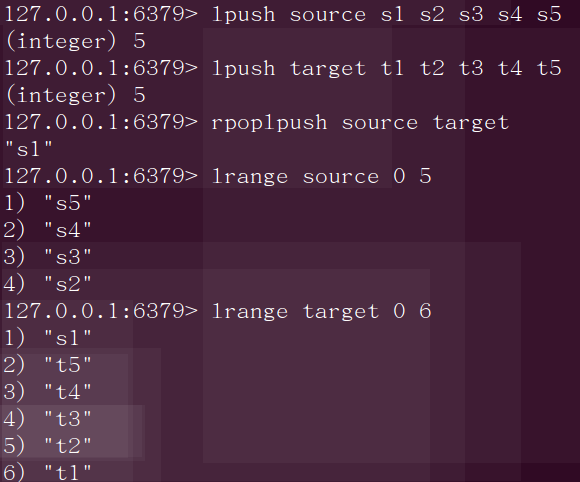
向列表中插入元素linsert key before|after pivot value。从左向右查找值为pivot的元素，然后根据before等决定插入位置，填写value数据。



## 5.9将数据从a列表转移到b列表

将元素从一个列表转移到另一个列表中，rpoplpush source target。还有其他类型的操作，比如lpoprpush等。

作用:source中rpop数据，然后lpush元素到target中.



S1:source [s5 s4 s3 s2 s1]

S2:target [t5 t4 t3 t2 t1]

S3:[s1 t5 t4 t3 t2 t1]

## 5.10模拟队列和栈操作

当队列使用时使用lpush和lpop或rpush和rpop操作；当成队列使用，使用lpush和rpop或者rpush和lpop.

## 5.11list-hash-string

|  |  |  |  |
| --- | --- | --- | --- |
| 操作 | list | hash | string |
| 单赋值 | lpush、rpush | hset | set |
| 多赋值 | lpush、rpush | hmset | mset |
| 单取值 | rpop、lpop、lindex | hget | get |
| 多取值 | lrange key start stop | hmget | mget |
| 数据长度 | llen key | hlen key | strlen key |
| 删除数据 | lrem key count value | hdel key fi |  |

# 6.set

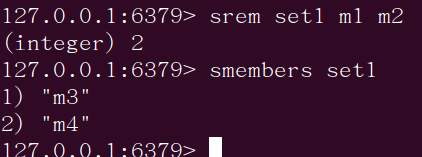
什么是集合?一堆数据放在一起就是集合。集合基础是hash table,时间复杂度都是o(1)。集合优势在于多个集合集进行并集、交集、差集操作。

## 6.1基本命令

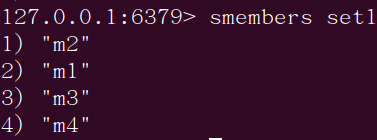
增加元素sadd key member[member ...],返回值是添加成功元素个数。



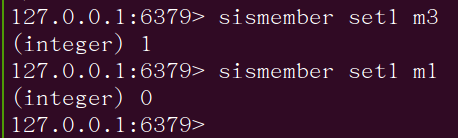
删除元素srem key member[member ...],返回删除元素个数



查看所有数据 smembers key



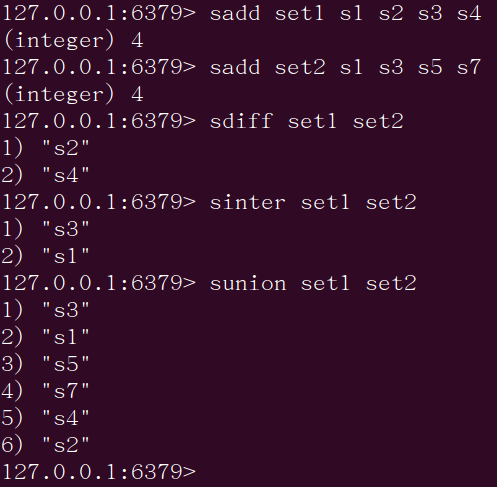
判断元素是否在集合中 sismember key member有返回1,没有返回0。



差集运算sdiff key [key ...]

交集运算sinter key [key ...]

并集运算sunion key [key ...]



应用场景：对数据顺序没有要求,且要求数据保持唯一，比如商品或者文章加标签操作,重复的标签不会添加。

## 6.2其他命令

获取集合中元素个数scard key

差集结果存储sdiffstore target key[key ...],将差集保存在target中。

交集结果存储sinterstore target key[key ...],将交集保存在target中。

并集结果存储sunionstore target key[key...],将并集结果保存在target中。

获取集合中随机元素srandmember key [count],并非严格意义上的随机获取。

从集合中弹出一个元素spop key。

# 7.zset

需要了解有序集合的数据结构，但是我并不了解有序结合，没有自学过这个数据结构，等我研究过后再详细地补上。

zset相关信息如下：

## 7.1.list-set-zset

三者存储空间都是2的32次方-1。list有序,set无序,zset有序。list不唯一,set唯一,zset唯一。

list数据结构基础是链表,两边添加和删除是优势,查询中间数据是短板;set数据结构是hash table,读取数据时间复杂度为o（1）;zset数据机构是hash table和skip list，读取数据时间复杂度为O(log(N))

有序集合比list更占用内存.

## 7.2.基本命令

添加元素 zadd key score member[score member...]将member中score值添加到有序集合key中;

score必须是整数或者浮点数,返回值是添加到有序集合中个数;member中score存在则更新,不存在则添加,新增返回1,修改返回0。

获取member的score数据,zscore key member;

返回指定索引区间的member, zrange key start stop [withscores],按照score从小到大排序.zrevrrange和zrange 的排序相反而已;

获取指定score区间的member, zrangebyscore key min max [whitscores] [limit offset count] ,-1表示最后1个,+inf和-inf表示正无穷和负无穷;

增加某个member的score值, zincrby key increment member,然后返回修改后的score;

Redis小结：如果key不存在或field不存在,那么一般先创建后进行操作;如果该操作不是该数据类型的,那么会报错;zset没用过,不知道具体使用什么场景.

## 7.3.其他命令

获取key中member个数,zcard key

获取指定score范围内的member个数 zcount key min max

删除1个或者多个member, zrem key member[member ...]

删除单个数据返回删除的元素,批量删除则返回删除的个数,包括不存在的member计算在内

按照索引范围删除元素 zremrangebyank key start stop,从索引0进行排序

按照score范围删除member zremrangebyscore key min max,删除范围正常是闭区间,如果不包含那么使用（表示开区间.

实在不熟悉有序集合数据结构，如果谁比较擅长，那么请重新编辑zset这章。

# 8.redis事务(理解)

事务中操作要么全执行，要么全都不执行。

## 8.1事务语法

事务定义：

multi

command1

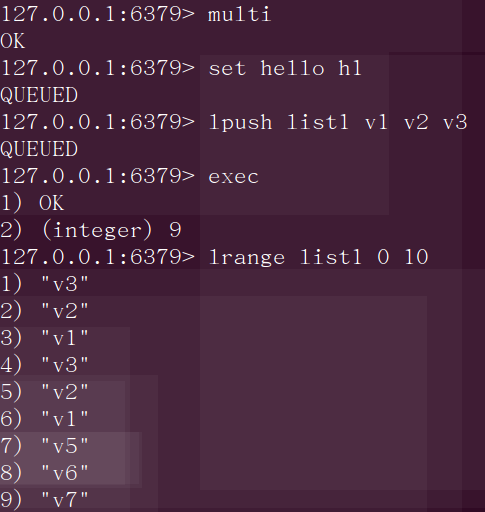
......

commandN

exec

语法解释如下：

1. multi告诉redis以下操作要执行的是事务，输入command1，redis返回queued表示命令已经加入队列，输入exec告诉redis事务输入完成，可以进行执行。
2. 在输入命令过程中断线了，那么事务不会执行.
3. 输入完exec后断线，还会执行事务，因为redis已经记录完要执行的命令。

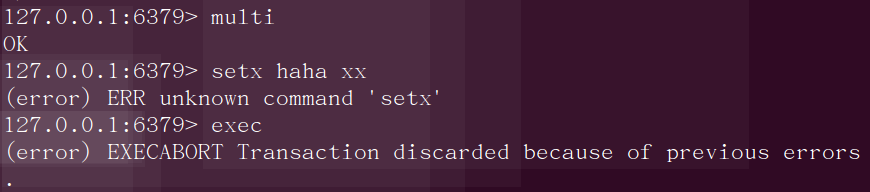


## 8.2错误处理

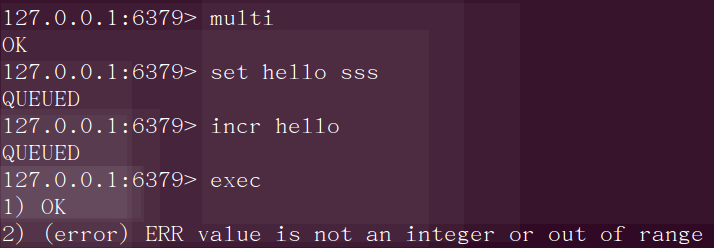
操作数据库时发生错误，事务就会回滚，redis如何处理呢？

1. 语法错误. redis事务执行过程发生语法错误，那么redis不会执行事务。像java编译时有错不执行。
2. 运行时错误. 在写事务时语法无错但运行时发生错误，正确的指令执行，错误的指令不执行，不会发生事务回滚

语法错误案例：



运行时错误案例：



如何规避错误，语法错误开发时可以找到，运行时错误根据合理命令规避风险。

问题如下：使用get获取数据，然后incr加以，然后set，对于不同的操作都进行这个过程，就无法保证数据的有效性。

解决方案：使用get获取数据时，防止其他操作修改数据，直到方法执行完，其他操作才可以修改。和java中上锁类似。

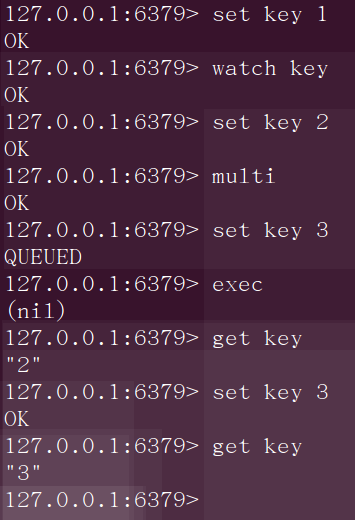
## 8.3 watch

下面是理解watch作用的关键。

1.watch监控多个key，一旦某个key不在事务中被修改，修改之后的第一个事务不执行，一直持续到第一个事务exec命令后，才会继续执行。

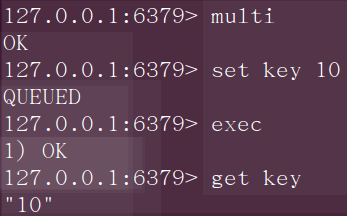
2.watch监控多个key，如果某个key在事务中被修改，那么事务是被执行的，后续的事务正常执行[2本质上是多余的]

案例：



可见被watch后的key在紧随的第一个事务中被修改是不执行的。

案例：



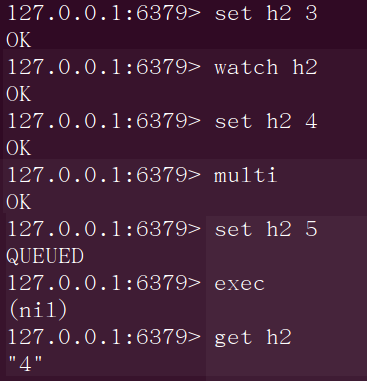
和上面的案例不同在于，被watch后的key执行exec后返回nil，此处返回的是ok。

可以如此理解：被监视的官员虽然有自由行动的权力，[但实施政令的权力已经被剥夺，在被监视其间所有的命令都是无效的]，只有在被解除监视的时候，官员才有继续实施政令的权力。

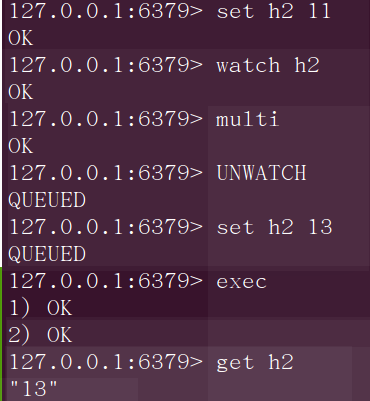
## 8.4 unwatch

取消监视使用unwatch

watch案例：



unwatch 案例：



如果没有修改数据，那么使用在事务中使用unwatch，数据修改可以进行事务存储；

如果在watch过程中修改数据，那么unwatch不会发生作用。

此处是测试验证的结果，并非来自书籍理论，等研究源码时再进行进一步确认，可做了解。案例请查看截图操作。



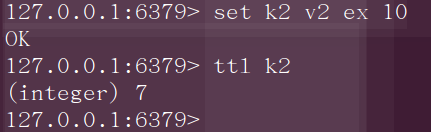
# 9.失效时间(掌握)

为何设置失效时间呢？药品设置失效时间的意义都知晓，吃了失效时间的药品可能会出人命；缓存的失效时间价值在于到失效时间，那么缓存就会被清除；如果数据量少的缓存也许设置失效时间无所谓，但是如果是大型网站海量缓存的话，如果不设置失效时间，那么批量的无用缓存会占用浪费大量的内存空间，为了保证内存的有效使用，比如失效的数据、过期的活动、验证码等数据过期就需要清除。

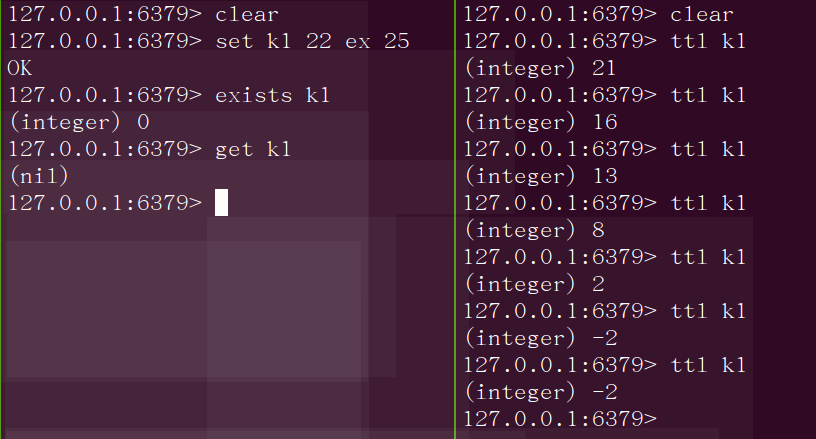
## 9.1使用场景

有时效的缓存，比如优惠活动、缓存、验证码、网站限时活动。有时效性的缓存必须设置失效时间。防止业务错误调用，节省内存空间。

设置某个key的时效时间即可，一般在设置缓存时进行失效时间设置。

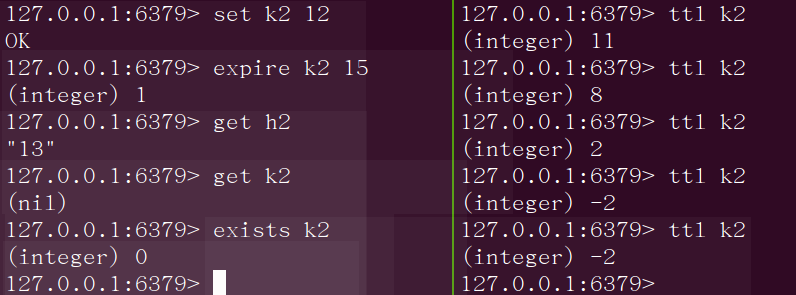


## 9.2失效设置



在设置key数据时可以设置失效时间，是可选项。ex 表示的就是失效秒数。ttl表示查看失效剩余时间。

语法：expire key seconds [seconds表示秒]

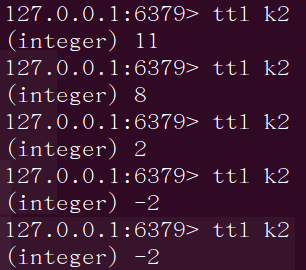




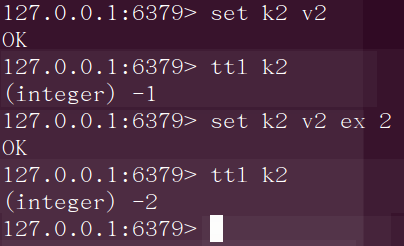
返回1表示设置成功，返回0表示设置失败或者key不存在

## 9.3查看失效剩余时间

使用ttl key查看剩余时间（秒数）



返回-2表示key已经过期不存在。

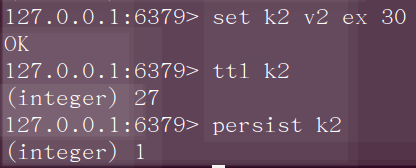


如果返回-1表示没有设置失效时间，即永久存在。

## 9.4取消失效时间

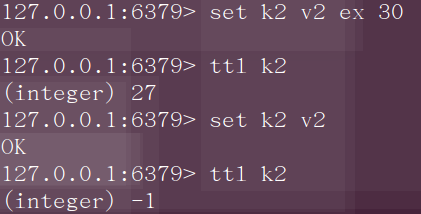
取消失效时间设置，可以使用以下3种方法：

1. persist key

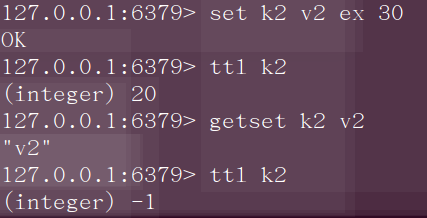


返回1表示清除成功，返回0表示失败或已经永久存在

1. set key



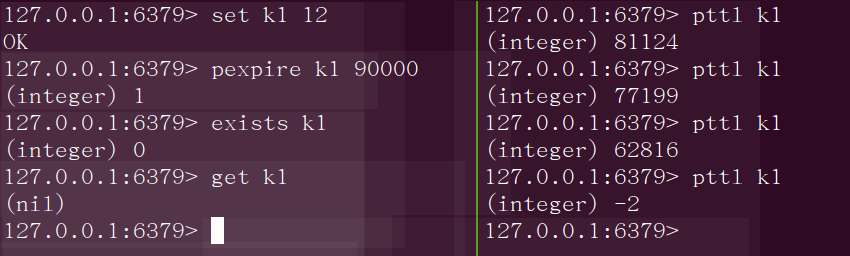
1. getset key



提示内容：在c中非0表示true，0表示false；在redis执行结果中，成功返回1，失败返回0，本身已经存在也返回0。

## 9.5失效命令对比

|  |  |
| --- | --- |
| 失效命令 | 查看剩余失效时间 |
| expire key second[秒] | ttl查看expire |
| pexpire key second[毫秒] | pttl查看pexpire |



## 9.6访问频率限制

有缺陷的方式：设置key失效时间，用户访问incr（key），如超过100次，没有失效，不允许用户继续访问。

key失效时间后重新创建key，进行限制。创建key却没设置成功失效时间，需要手动处理该key，且并非有效的时间段。

合理解决方案：

S1：给key设置list，数据不超过10条，继续访问；

S2：存在10条，比较要插入数据和存在数据的10条数据写入时间，小于特定时间则不插入，大于特定时间则插入；

S3：删除10条数据前的数据

## 9.7设置redis使用最大内存

问题发生情景：限制redis最大占用内存，同时按照一定规则淘汰不需要的缓存key。

[参考Redis配置文件详解文档中内存策略]

1. 修改配置文件的maxmemory参数限制可用最大内存;

2.超过最大内存，根据maxmemory-policy参数指定删除策略

3.删除策略有6，常用allkeys-lru

|  |  |
| --- | --- |
| volatile-lru | 使用LRU算法删除设置过期时间的key |
| allkeys-lru | 使用LRU算法删除任何key |
| volatile-random | 随机删除设置过期时间的key |
| allkeys-random | 随机删除任意1个key |
| volatile-ttl | 删除即将到期的key、辅以ttl |
| noeviction （默认） | 不删除任何key，仅返回一个写错误 |

# 10.mq(掌握)

消息队列实现方式如下：

1.使用list数据结构实现点对点队列

2.使用redis自身提供的订阅发布实现订阅发布队列

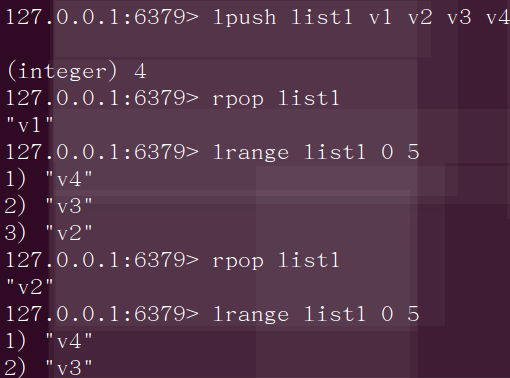
## 10.1点对点实现

方式1：lpush和rpop或rpush和lpop

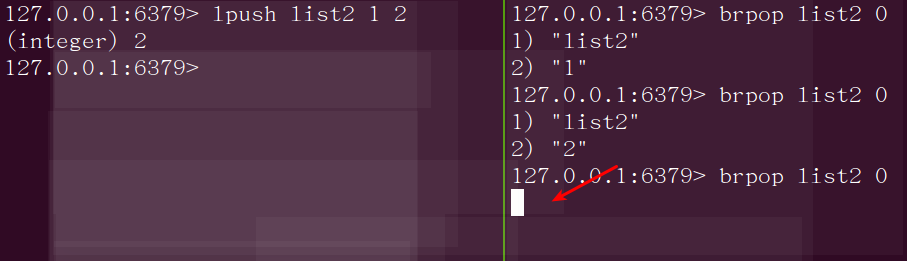
方式2：blpushbrpop或brpush和blpop

方式1和方式2对比:方式1不知道队列中什么时候有数据，所以需要你不停去问快递人员有没有我的快递，有就拿走，没有我也去问；方式2一直在快递人员那等着，有我的快递我就拿走，没有我也等着；方式1和方式2根本差异在于列表中没有元素时，brpop和blpop会一直阻塞连接，直到有新元素加入。

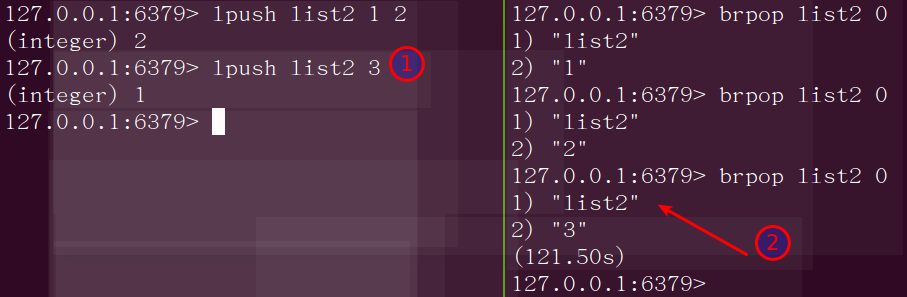
方式1：



方式2：brpop说明：brpop key time， key表示键名，time表示超时时间（秒），0表示无限制等待.



红色箭头处进行闪烁，等待list2中左边输入的数据。



在1处左侧推入数据，会在2处自动弹出数据。

方式1和方式2使用时，优先使用阻塞式brpop等操作。

总结：redis的点对点队列，是list操作而已，没法根据路由进行数据分派，所以点对点队列功能比较差，适合点对点发送中消息比较简单的情况。

优先级队列,在我看来就是鸡肋般存在,在若干list中，brpop key[key...] timeout,优先获取左边的key的元素。

## 10.2发布/订阅实现

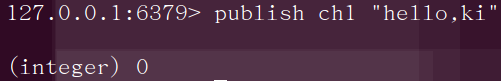
简单解说：订阅者可以订阅多个频道（channel），发布者向特定频道发送消息，所有订阅该频道的订阅者都可以收到消息。

类比生活场景：任何人向邮局订购报纸，那么邮局会向每个订阅者，每日都会向订购者发送报纸。邮局就是发布者，订购报纸的每个人都是订阅者。

## 10.3发布者

语法：publish channel message

返回值表示订阅该channel的订阅者个数

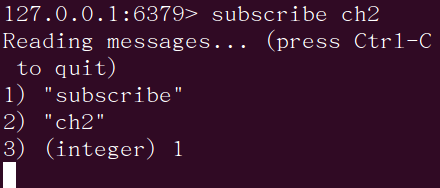


和使用list比较，redis的订阅发布模式不会进行消息持久化，如果在发送过程中消息丢失了，那么消息就彻底丢失。

从list的点对点模式和redis自身提供的订阅发布功能来看，和专有功能的消息队列相比还是相差比较大，redis的消息队列适合功能不太复杂的应用场景中。

## 10.4订阅者

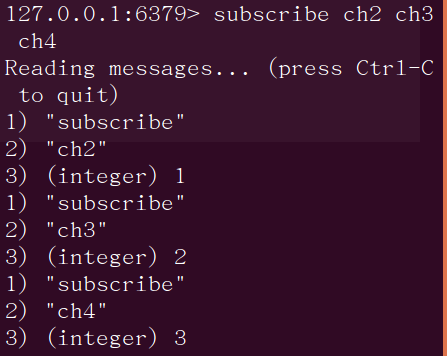
语法：subscribe channel[channel ...]



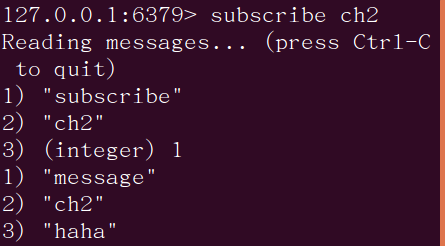
订阅某个频道返回值：

1. "subscribe"表示订阅成功；
2. "ch2"订阅频道名称；
3. (integer) 1当前客户端订阅成功个数

订阅多个channel



接收消息：



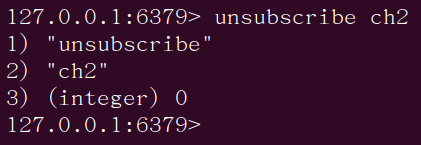
接收消息返回值：

1. "message"表示接收的是消息；
2. "ch2"表示channel名称；
3. "haha"表示消息内容

## 10.5取消订阅

取消订阅某个channel：

unsubscribe channel[channel ...]



取消订阅返回值：

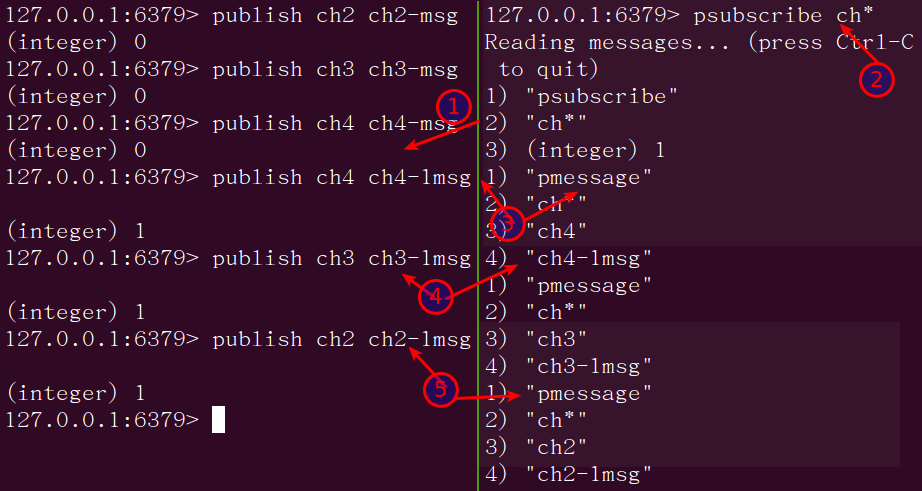
1. "unsubscribe"表示取消订阅；
2. "ch2"表示取消订阅channel名称；
3. (integer) 0表示当前订阅channel个数

## 10.6规则订阅

指定规则进行订阅： psubscribe channel.\*?

规则和keys glob风格一样:

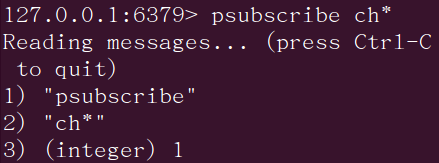
\*表示任意个数，？表示单个，[]表示匹配[]中任意某个字符



说明内容如下：

1. 其中ch2、ch3、ch4是创建发布3个channel；
2. 使用的是规则订阅模式，同时订阅1中的3个channel；
3. 其中ch4渠道发布消息，订阅的渠道接收消息。ch3、ch2以此类推。

其中订阅消息返回值：



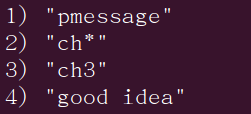
其中：

1)"psubscribe"表示使用规则进行订阅成功

2)"ch\*"，表示订阅规则

3) (integer) 1，表示订阅成功个数。

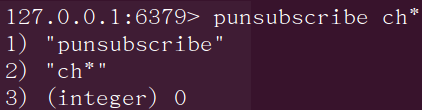
接收消息：



1. "pmessage"表示指定规则接收消息；
2. "ch\*"表示消息接收规则；
3. "ch3"收到消息的channel；
4. "good idea"表示收到的消息。

## 10.7规则退订

punsubscribe [pattern],根据规则进行退订，没有参数表示退订全部。



1）"punsubscribe"，表示按照订阅规则退订

2) "ch\*"，表示退订规则

3) (integer) 0，表示目前剩余订阅频道个数

总结：redis的订阅发布模式，由于在发布的过程中数据一旦丢失不能进行恢复，对于重要的数据就不可以采用redis的订阅发布模式。对于不重要的数据，丢失不丢失无所谓，那么可以使用redis的订阅发布模式；对于redis的点对点功能，适合功能比较简单场景。对于复杂的点对点功能而言，不太建议使用redis的list消息队列。

# 11.pipeline(理解)

## 11.1为何使用管道

为什么使用管道？使用管道和使用sql批处理作用类似。redis使用tcp协议，向redis服务器发送命令和返回给客户端都需要时间，叫往返时间延迟。

案例:去商场买苹果、衣服、手机，通常一次性购买完，如果分3次购买，那么在路上耗费3次来回时间。

使用管道的目的在于减少发送命令和返回命令的时间延迟.使用管道可以一次性发送多条命令，并执行完一次性返回。

注意：管道中的命令之间不相互依赖。

## 11.2节省空间

首先要明白redis直接运行在内存上；其次不要为牺牲可读性来精简键名来达到节省空间目的；如果数据具有时效性，那么加上失效时间。

# 12.持久化(理解)

使用redis时，数据会从内存持久化到本地硬盘中，重启后数据从硬盘上加载到内存中，从而达到数据不丢失的目的。

持久化方式：RDB方式和AOF方式，可单独使用，可结合使用。

RDB是redis默认的持久化方式。

## 12.1.RDB模式

RDB模式，是Redis Data Base的简称。是通过快照snapshot完成，当符合某些条件时redis会自动将内存中所有数据进行快照并保存在硬盘中。

[请参考redis配置文件详解中5RDB模式配置来理解下面内容。]

快照条件可自定义配置，参数有2:时间和改动键个数，指定时间内被更换的键的个数大于指定数值时进行快照.

配置文件中内容如下：

save 900 1

save 300 10

save 60 10000

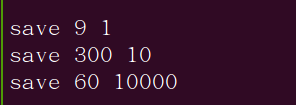
以上三个配置是或的关系，比如900秒内至少有1个键被改动后进行持久化.

禁用快照，删掉所有save即可.

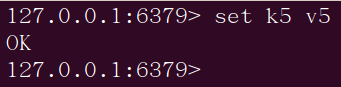
redis将快照默认存储当前目录dump.rdb文件，可以配置dir和dbfilename参数来指定快照文件的存储路径和文件名.

查看效果：将900秒内至少有1个改动就快照修改为9秒内有1个改动就快照。

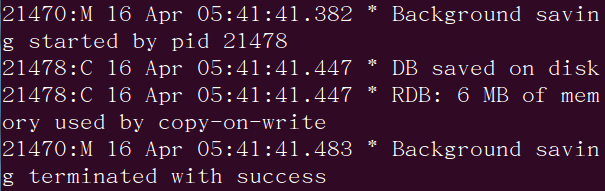
1.配置文件修改如下，此配置无法进行热配置修改，修改后需要重启Redis



1. 输入修改命令



1. 服务器操作输出



基本上是操作数据后，及时进行持久化操作。

## 12.2.RDB快照过程

S1：redis使用fork函数复制一份当前进程（父进程）的副本到子进程；

S2：父进程继续接收处理客户端的命令，子进程开始将内存中的数据写入到硬盘中；

S3：当子进程写入完所有的数据后会用该临时文件替换旧的rdb文件，至此一次快照完成；

当执行S1时中fork函数，发生那刻父子进程共享内存数据，当父进程更改某片数据，系统会将该片数据复制一份以保证子进程不受影响；当执行S2时新的rdb文件存储是执行fork函数一刻的内存数据。

## 12.3.RDB总结

1. rdb是redis database的缩写；
2. rdb进行快照过程不会修改旧rdb文件，会将旧文件替换掉；
3. 通过定时备份rdb文件来实现redis的备份
4. rdb是压缩后的二进制文件，便于传输
5. 使用save或bgsave手动执行快照，前者由主进程执行快照会阻塞其他请求；后者通过fork子进程进行快照操作。
6. redis启动后会读取rdb文件，将数据从硬盘加载到内存中
7. 1G文件加载到内存中花费时间大约20-30分钟

使用rdb持久化缺点：当redis异常退出就会丢失最后一次快照后的数据，而最后完整的数据是最后一次成功快照后的数据，在此之后的数据全部会丢失。

## 12.4.AOF模式

AOF模式是append only file简称。由于RDB模式下redis发生异常会丢失最后1次快照后的数据，然而很多重要业务是不允许丢失数据的，那么如何处理呢？

## 12.5AOF总结

[请参考redis配置文件详解中AOF模式配置]

AOF模式会将每条更改的redis命令写入硬盘aof文件中;aof文件保存位置和rdb文件相同;默认文件名是appendonly.aof，可通过appendfilename修改;默认情况下开启RDB模式，而非AOF模式，修改appendonly yes;在redis3.2.8中没有见到appendonly.aof.

aof是纯文本文件,记录操作命令记录;启动redis会逐个加载aof文件的命令将硬盘数据加载到内存中，速度较rdb慢;操作系统缓存机制，数据先进入硬盘缓存，默认30秒会执行同步操作，写入硬盘中。如果30秒内出现异常，数据会丢失，要求redis写入aof文件后将缓存内容同步到硬盘中;appendfsync always，表示每次执行写入都会执行同步，最安全但是最慢;appendonly everysec，默认规则，每秒执行1次同步操作;appendonly no，从不执行同步操作，有系统执行同步操作; 一般使用默认即可，兼顾性能和安全;允许同时开启rdb和aof模式，兼顾数据安全和备份，重启后会使用aof来恢复文件.

# 13.复制(理解)

## 13.1复制意义

意义1：数据不丢失

在单实例redis上，如果硬盘发生故障，那么数据丢失；

某实例redis上数据更新，自动更新数据到其他redis服务器上，redis提供复制实现自动同步，从而数据不丢失情况。

意义2：在复制基础上实现数据读写分离。

## 13.2复制配置

同步数据库分为主(master)数据库，从(salve)数据库;主数据库进行写操作，发生写操作自动讲数据同步给从数据库;从数据库一般执行只读操作，接收主数据库同步过来的数据;一般是1主多从的配置.

## 13.3如何实现复制

方式1：直接在命令行中进行复制

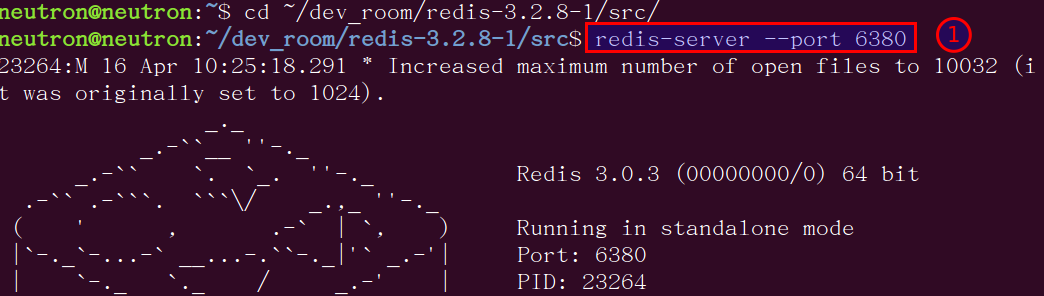
参考命令如下：需要注意命令格式

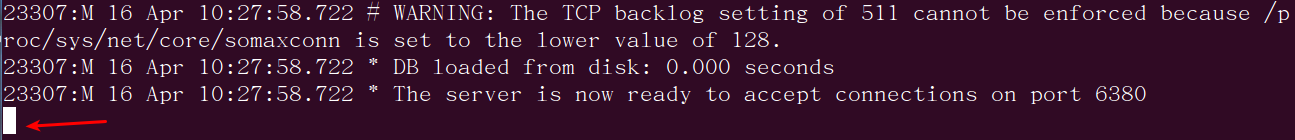
master：redis-server --port 6380

slave： redis-server --slaveof 127.0.0.1 6380

操作步骤：

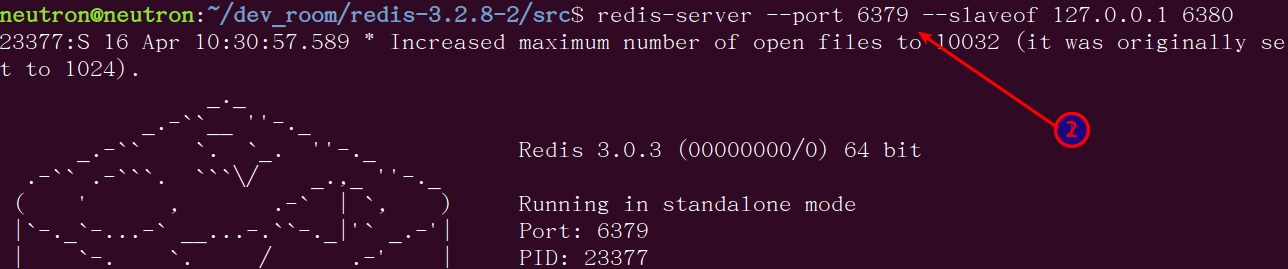
1开启master服务器：redis-server --port 6380



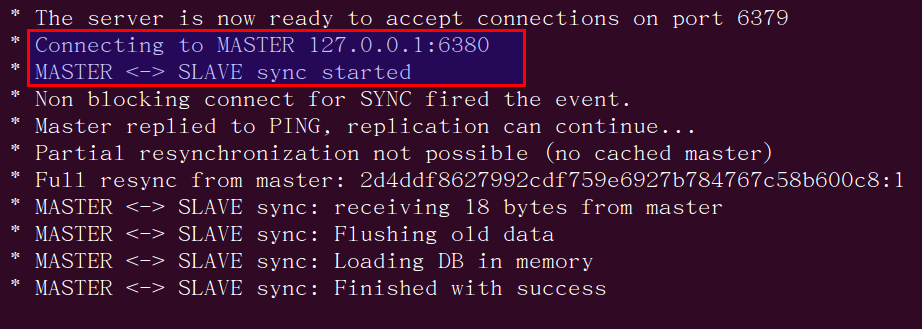


没有任何从服务器连接

2开启slave服务器：redis-server --port 6379 --slaveof 127.0.0.1 6380

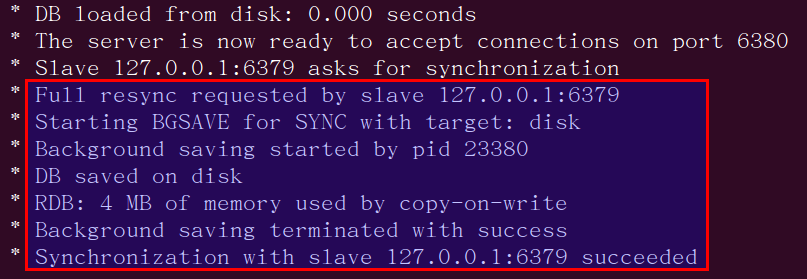


末尾提示语：



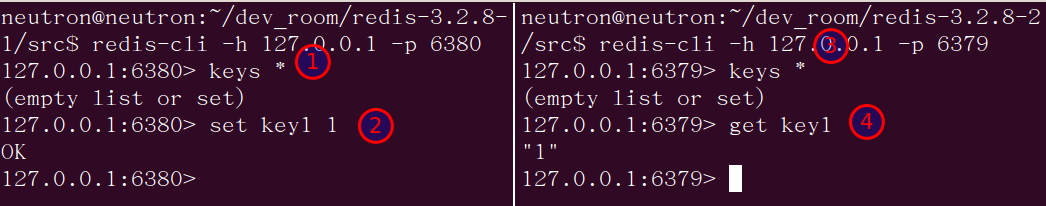
此时已经和master建立起主从关系。

1. master提示语

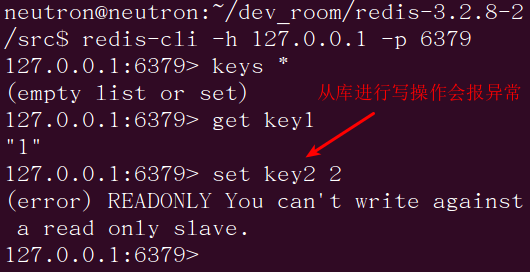


提示已经和slave建立起主从关系

4.客户端操作



在主库操作添加数据，那么自动同步到从数据库。



默认从数据库只读，进行写操作会报错。如果从数据库如何进行写操作，从数据库配置slave-read-only no即可。从数据库更新不会更新到主数据库，但是主数据库更新会覆盖从数据库数据的改动。

如果已经是某个数据库的从库，那么会停止和旧主数据库的关心，同步新主数据库。使用slaveof no one使当前从数据库转化为主数据库

只有学习时候才会在命令行中进行主从设置的，在生产环境中不会在启动时设置主从关系，只会在从数据库的配置文件中进行配置。

Q1:在配置文件中如何实现复制功能？

在从数据库配置文件中加入 slaveof 主数据库ip 端口即可，主数据库不需要任何配置.

## 13.4.复制原理

复制过程：

1.salve启动会向master发送sync命令

2.master接收sync命令在后台进行rdb持久化，保存期间的收到的命令缓存

3.快照完成，将所有快照和缓存命令发给salve数据库

4.slave载入快照并执行收到的缓存命令

5.不支持断点续传

6.执行复制操作，就会执行rdb持久化，快照操作

7.slave数据库可以作为master数据库，然后有salve数据库关联。

## 13.5读写分离

读写分支的基础是主从复制。读写分离意义在于单机无法应付大量读操作，需要通过复制建立slave数据库，master只负责写，salve负责读。

## 13.6.从库持久化

从库持久化的目的在于提高性能，主数据库不进行持久化，对某个从数据库进行持久化。master禁用持久化，从数据库崩溃，可以通过主数据库进行数据同步。master崩溃，可以将salve数据库转化为master数据库，原来的master变成salve数据库。

# 14.安全(掌握)

可信任环境，redis的安全设计是以redis运行在可信任环境前提。不允许外界直接连接在生产环境的redis服务器，而是通过应用程序中转连接redis服务器。

运行在可信任环境是保证redis安全的最重要保障。redis默认配置会接收来自任何地址发送来的请求。

## 14.1绑定安全IP

如何设置只来自某个机器连接，比如只有本机，使用bind参数。比如 bind 127.0.0.1即可,bind参数可以绑定多个地址，中间使用空格间隔.更多设置可以从防火墙规则设置。[具体请参考redis配置文件详解中网络连接]。

## 14.2数据库密码

配置文件中requirepass设置redis连接密码、重启redis服务器、设置密码后连接方式使用auth [password]。

Eg：

1.在配置文件中修改redis.conf

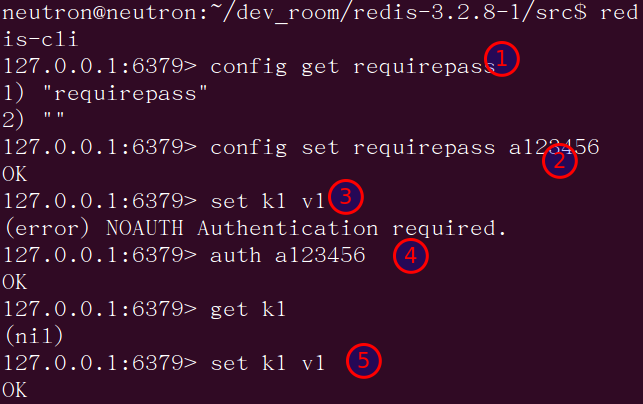


2.重启redis server

3.使用auth password

网上大多推荐此方法，但是我本地不起作用，参考：http://blog.csdn.net/lxpbs8851/article/details/8136126

在客户端进行设置



1.config get requirepass 查看目前redis服务器设置的密码

2.config set requirepass a123456，给当前redis服务器设置密码；

3.设置密码后，创建key-value，提示需要验证

4.验证登录，auth password

5.验证登录后，正常使用redis服务器。

问题:在测试过程中重启redis服务器和redis-cli后，设置的密码继续为nil，没有起到任何作用，不知道什么原因。

redis是单线程模型，且性能极高，攻击者可以穷举密码，所以密码要足够复杂；redis复制时master设置密码，需要在从库的配置通过masterauth参数设置master密码，那么连接从库自动使用master密码。

## 14.3.命令重命名

对于及其危险的命令，redis支持配置文件中将命令重命名。

|  |  |
| --- | --- |
| flushall | 删除所有数据库数据 |
| flushdb | 删除当前数据库数据 |

[请参考redis配置文件中安全-命令重命名]

1.在配置文件中进行修改，重命名flushall：

rename-command flushall aliasname

1. 禁用某个命令 rename-command [command] [''],最后是空字符串。

rename-command flushall ""

# 15.通讯协议(了解)

redis支持两种通讯协议，二进制安全的统一请求协议(unified request protocol)和简单协议。两者命令格式不同，但是返回值格式一致。

关于通讯协议简单了解即可。

## 15.1简单协议

1.适合telnet程序和redis通信

2.终端输入的命令展示都是采用简单协议，比如set foo bar，采用命令 + 参数形式

3.返回类型

|  |  |  |
| --- | --- | --- |
| 错误回复 | -开头 | -ERR |
| 状态回复 | +开头 | +OK |
| 整数回复 | :开头 | :3 |
| 字符串回复 | $开头 字符串长度 | $5,nil返回$-1 |
| 多行回复 | \*开头 字符串回复组数 | \*4 |

## 15.2.统一请求协议

参考15.1的返回值类型

以set foo blar为例统一请求写法：

\*3\n\r\\n$3\r\\nset\r\\n$3\r\\nfoo\r\\n$4\r\\nbar

\*3表示字符数组长度3,即[set food blar是字符数组的3个元素]

$3表示字符串长度是3，即set的长度！之后内容依次类推。

统一请求协议使用场景：

1.AOF文件和主从复制使用统一请求协议

2.开发redis客户端推荐使用统一请求协议

3.通过telnet向服务器发送命令使用简单协议即可。

# 16.redis客户端介绍

## 16.1查看耗时命令(掌握)

自带客户端工具redis-cli，查看耗时命令 slowlog get来查看当前耗时命令日志;在配置文件sowlog-log-slower-than 参数，单位微秒，1000000=1秒，默认10000;通过配置slowlog-max-len限制记录记录条数.

Eg:

slowlog get

1)(integer) 4 表示该日志唯一Id

2)(integer) 13456064132 表示执行命令的unix时间\n

3)(integer) 59,表示命令执行耗时时间，单位微妙\n

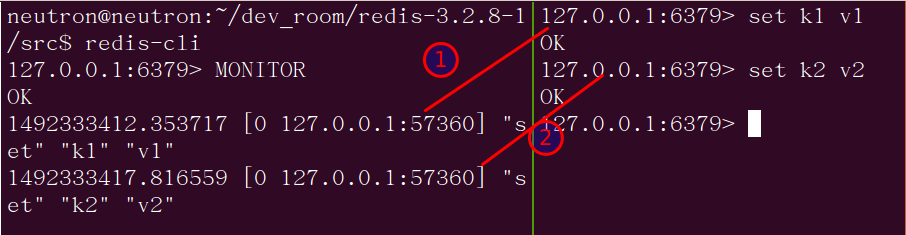
4)1)"get" 表示命令

2)"foo" 表示参数

记录所有命令:slowlog-log-slower-than设置为0;设置为负数表示关闭耗时命令日志.

## 16.2命令监控(掌握)

命令监控,命令monitor,任何执行命令都会在redis-cli打印出来.monitor影响性能，适合调试和纠错.案例如下：



## 16.3phpRedisAdmin(了解)

可树型查看键列表、依赖php的redis客户端predis、获取键列表使用keys \*，因此生产环境不建议使用。

## 16.4RDBtools(掌握)

redis的快照分析器，根据快照导出json数据文件,进行分析;根据python开发;如果没有快照，使用save进行手动生成快照;将快照导出成json格式命名：

rdb --command json /path/to/dump.rdb > output\_filenam.json

生成空间使用报告：

rdb -c memory /path/to/dump.rdb > output\_finame.cvs

报告字段说明:

1>database 存储该键的数据库索引

2>type 键类型,type key

3>key key name

4>size\_in\_bytes 键大小

5>num\_elements key元素个数

6>len\_largest\_elements 最大元素长度